

Contents

Glossary	4
1 Introduction	6
1.1 Background	6
1.2 Key Concepts	6
1.2.1 Voting	6
1.2.2 Blockchain technology	7
1.3 Problem & motivation	8
1.4 Purpose & delimitations	9
1.5 Document structure	10
2 Technical background	11
2.1 Ethereum Blockchain	11
2.1.1 Accounts	12
2.1.2 Transactions	13
2.1.3 Gas	14
2.1.4 Consensus mechanisms	15
2.1.5 Dapps	16
2.1.6 Smart contracts	17
2.2 Web Applications	17
2.2.1 Structure	17
2.2.2 Business use	18
2.2.3 Development	19
3 Implementation	20
3.1 Tools	20
3.1.1 React JS	20
3.1.2 Web3.js	20
3.1.3 Ganache	21
3.1.4 MetaMask	22
3.2 System Design	22

3.2.1	Architecture	23
3.2.2	Diagrams	25
3.3	Results	27
3.3.1	Admin dashboard	27
3.3.2	End-user (voter) application	29
3.3.3	Smart contract	33
3.3.4	System properties	36
4	Conclusion and perspectives	37

List of Figures

1.1	Abstract Blockchain blocks visualization	7
1.2	The Votomatic vote recorder, a punch card voting machine originally developed in the mid 1960s	8
1.3	Proof-of-concept "Verum" Logo	9
2.1	Ethereum Blockchain logo	11
2.2	Abstract Ethereum accounts visualization	13
2.3	Abstract representation of an Ethereum transaction	13
2.4	Diagram of the role of gas in Ethereum [13]	14
2.5	Diagram of centralized and decentralized applications topology	16
3.1	React JS framework logo	20
3.2	Web3 logo	21
3.3	Ganache Logo	21
3.4	Ganache GUI	21
3.5	MetaMask Logo	22
3.6	the architecture of a client application interaction with a blockchain network using Web3 and MetaMask	22
3.7	High level architecture of the proposed system and the interaction between its components	23
3.8	Sequence diagram of the process of creating and launching a new voting event	25
3.9	Sequence diagram of the process of casting a vote	26
3.10	Election overview tab of the admin dashboard	28
3.11	Event creation tab of the admin dashboard	28
3.12	Logs tab of the admin dashboard	29
3.13	Voter application authentication screen	30
3.14	Choices presentation screen	31
3.15	Choice submitting screen	31
3.16	Vote casting feedback screen	32
3.17	Voting smart contract, written in solidity language	33

Glossary

51% attack A 51% attack refers to an attack on a Proof-of-Work (PoW) blockchain where an attacker or a group of attackers gain control of 51% or more of the computing power or hash rate. PoW is a system of consensus used by blockchains to validate transactions.. 15

cryptocurrency A cryptocurrency, crypto-currency, or crypto is a digital asset designed to work as a medium of exchange wherein individual coin ownership records are stored in a ledger existing in a form of a computerized database using strong cryptography to secure transaction records.. 11

Cryptoeconomics Cryptoeconomics describes an interdisciplinary, emergent and experimental field that draws on ideas and concepts from economics, game theory and related disciplines in the design of peer-to-peer cryptographic systems.. 15

ETH ETH is a cryptocurrency. It is scarce digital money that you can use on the internet, It's the currency of Ethereum apps.. 15

Proof-of-concept also known as proof of principle, is a realization of a certain method or idea in order to demonstrate its feasibility, or a demonstration in principle with the aim of verifying that some concept or theory has practical potential. A proof of concept is usually small and may or may not be complete.. 9

Abstract

This research aims to explore one very prominent and potential application for the blockchain, which is a decentralized electronic voting system, by creating a proof-of-concept of a voting application, capable of launching an election, casting votes and displaying results, all while ensuring transparency, anonymity, security and above all the correctness of the results.

Keywords: Decentralized Electronic Voting System, Voting, Blockchain

Chapter 1

Introduction

1.1 Background

As the hype surrounding bitcoin[17] is slowly waning, the industry is becoming more and more interested in its underlying technology: the blockchain. Now, proponents claim blockchain technology to be one of the most important new technologies of our time, about to take society by storm, and bitcoin served as the first, most thorough and complete proof of concept. In fact, Marc Andreessen, founder of VC firm Andreessen Horowitz and one of the most influential members of Silicon Valley, claimed in a New York Times article that the invention of the blockchain is as important and influential as the creation of the Internet itself[15].

Along with assets registry, secure sharing of medical data and supply chain and logistics monitoring, electronic voting is one of the most prominent potential applications for the surging blockchain technology.

1.2 Key Concepts

1.2.1 Voting

According to Oxford English dictionary, voting is *"a formal indication of a choice between two or more candidates or courses of action, expressed typically through a ballot or a show of hands"*, and the very first forms of voting date back to approximately 508 B.C in ancient Greece where the earliest form of democracy was implemented[8]. Greeks had a "negative" election – that is, each year voters, who were the male landowners, were asked to vote for the political leader or "candidates" they most wanted to be exiled for the next ten years.

The early ballot system was voters wrote their choice on broken pieces of pots, ostraka in Greek, and from this name comes our present word to ostracize. If any "candidate" received more than 6,000 votes then the one with the largest number was exiled. If no politician received 6,000 votes then they all remained. Since voters were only male landowners, the number of voters was small. If there was a fairly even spread of votes, no one would be exiled, so usually, only very unpopular political leaders were ostracized or exiled.

The election is the backbone of modern democratic societies, it often takes place at a polling station; it is voluntary in some countries, compulsory in others, such as Australia.

1.2.2 Blockchain technology

Blockchain is a distributed database that maintains a secure and ever-growing ledger of records (known as blocks), allowing for the creation of a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system. A blockchain is essentially a digital ledger of transactions that is duplicated and distributed across an entire network of computers.

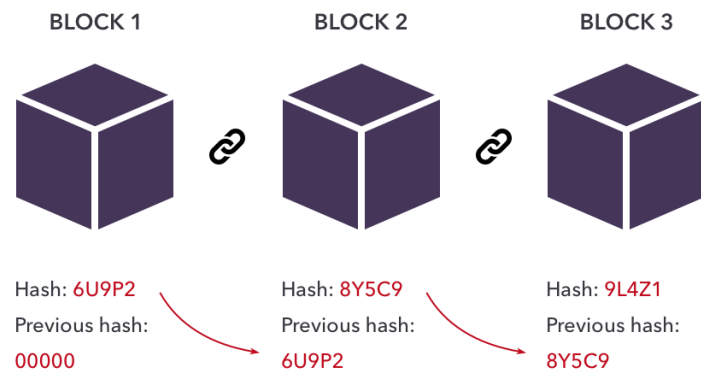


Figure 1.1: Abstract Blockchain blocks visualization

A blockchain network can track orders, payments, accounts, production and much more. And because members of the network share a single view of the truth, you can see all details of a transaction end to end, giving you greater confidence, as well as new efficiencies and opportunities.

1.3 Problem & motivation

Elections are expensive, complicated to set up and even on extreme cases fraudulent, this democratic system is almost 3000 years old and it has not evolved since ancient Greece. It entails an obligation for the voters to rely on third party officials and groups to safely collect votes and publish results, In the case of the modern democratic nation of France for example, there are only 3000 magistrates delegated to supervise and monitor more than 35 000 communes[1], a feat that is inconceivable for them to physically achieve, so setting the human malice apart still leaves room for human error, both leading to the voters carrying justifiable suspicions to any result the ballot may yield. Classic elections also carry hefty financial burdens, the 43rd general election (December 31st, 2020) of Canada had a total estimated cost of \$502.4 million[4], the equivalent of the cost of building 5 new hospitals, including administrative areas, operating and emergency rooms, and space for 120 beds[2].

Then comes technology, the premise held while entering various fields is that it will increase efficiency, cut costs and leave all parties involved satisfied, with an abundance of examples of when technology did also keep that premise, voting seems to be a logical place to land in next. Electronic voting systems and semi-electronic ones existed for quite some time, from punched card systems (see figure 1.2) to modern-day digital voting machines. However, they only served to optimize the physical act of punching a card or putting a piece of enveloped paper in a box, although that did cut some costs, may be made the experience of voting more pleasant, but it introduced more security concerns and most importantly it did not solve the essential problem of eliminating third parties, because people still had to trust the hardware and software engineers that programmed the machines to cast the votes, and the servers to not allow for any tampering with the data.

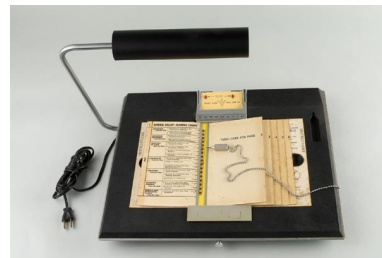


Figure 1.2: The Votomatic vote recorder, a punch card voting machine originally developed in the mid 1960s

1.4 Purpose & delimitations

The goal of this research is to investigate the possibility of using the new surging blockchain technology as the underlying technology powering an electronic voting system, offering the ability to store the results in a decentralized ledger that should grant the solution attributes like immutability and transparency. We intend to achieve our goal by designing a decentralized e-voting system as a Proof-of-concept, capable of launching an election, casting votes, and displaying results, all while ensuring transparency, anonymity, security and above all correctness of the results. It is of major importance to define the scope of this research document and its underlying implementation.

First, the implementation of the proof of concept is not by any means complete and thorough but instead functionality was compromised in favor of delivering an actual election on the blockchain.

Second, regarding blockchain technology, this document will not dive deep into the inner workings of the technology, topics like hash functions, cryptography and proof of work mechanism will not be presented in a manner that debates their efficiency since we judge the scientific literature on these subjects to be of abundance, the focus will instead fall on how the application of this technology on voting would look like. Also as Asaf Ashkenazi said *"Every device and system is hackable—it's just a matter of time and hacker motivation."*[3] so our focus will not fall on defending the robustness of our implementation or any future implementation but rather it will fall on highlighting the promise that this technology presents in terms of robustness. Lastly, code snippets that are unique to this project are included.



Figure 1.3: Proof-of-concept "Verum" Logo

1.5 Document structure

This document is presented in 4 chapters, starting with the introductory chapter in which we will present the reader with a bit of background of the topic and then delve into formally defining the problem we intend to tackle, followed by a brief description of what lies beyond the scope of this research.

In the second chapter, the reader will be presented with sufficient technical background about the two major fields involved in making this project come to life, being the Ethereum blockchain and Web applications. The next chapter will be about the implementation of our proof-of-concept, a chapter in which tools will be introduced and results will be exposed using diagrams, screenshots, and plain old language.

Finally, in the fourth and last chapter, the results and insights gained through the journey of making our proof-of-concept will be discussed, few conclusions drawn and perspectives on what could be enhanced moving forward with this project.

Chapter 2

Technical background

This chapter presents an overview of the technical concepts involved in the realization of this project, namely the Ethereum Blockchain and Web applications

2.1 Ethereum Blockchain

Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum, and its own programming language, called Solidity. As a blockchain network, Ethereum is a decentralized public ledger for verifying and recording transactions. The network's users can create, publish, monetize, and use applications on the platform, and use its Ether cryptocurrency as payment. Insiders call the decentralized applications on the network "dapps." [5].

Ethereum intends to create an alternative protocol for building decentralized applications, providing a different set of tradeoffs that its creators believe will be very useful for a large class of decentralized applications, with particular emphasis on situations where rapid development time, security for small and rarely used applications, and the ability of different applications to very efficiently interact, are important. Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.



Figure 2.1: Ethereum Blockchain logo

In the Ethereum universe, there is a single, canonical computer (called the Ethereum Virtual Machine, or EVM) whose state everyone on the Ethereum network agrees on. Everyone who participates in the Ethereum network (every Ethereum node) keeps a copy of the state of this computer. Additionally, any participant can broadcast a request for this computer to perform arbitrary computation. Whenever such a request is broadcast, other participants on the network verify, validate, and carry out ("execute") the computation. This causes a state change in the EVM, which is committed and propagated throughout the entire network.

2.1.1 Accounts

In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

- The **nonce**, a counter used to make sure each transaction can only be processed once
- The account's current **ether** balance
- The account's **contract code**, if present
- The account's **storage** (empty by default)

"Ether" is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees. In general, there are two types of accounts: **externally owned accounts**, controlled by private keys, and **contract accounts**, controlled by their contract code. An externally owned account has no code, and one can send messages from an externally owned account by creating and signing a transaction; in a contract account, every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

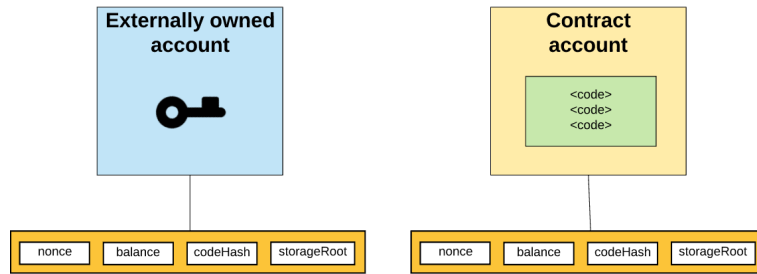


Figure 2.2: Abstract Ethereum accounts visualization

2.1.2 Transactions

An Ethereum transaction refers to an action initiated by an externally-owned account, in other words an account managed by a human, not a contract. For example, if Bob sends Alice 1 ETH, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction.

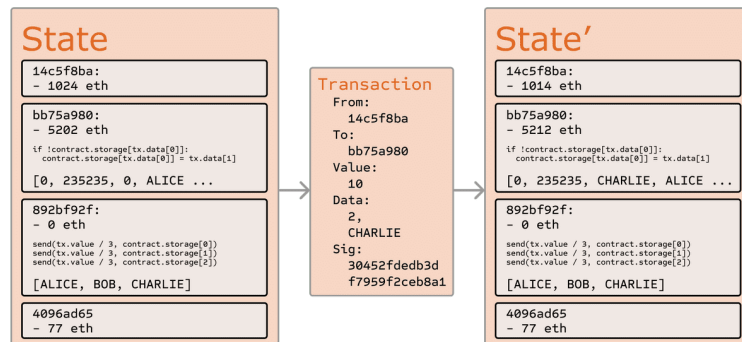


Figure 2.3: Abstract representation of an Ethereum transaction

Transactions, which change the state of the EVM, need to be broadcast to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will execute the transaction and propagate the resulting state change to the rest of the network.

Transactions require a fee and must be mined to become valid.

A submitted transaction includes the following information:

- **recipient** - the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)

- **signature** - the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorised this transaction
- **value** - amount of ETH to transfer from sender to recipient (in WEI, a denomination of ETH)
- **data** - optional field to include arbitrary data
- **gasLimit** - the maximum amount of gas units that can be consumed by the transaction. Units of gas represent computational steps
- **gasPrice** - the fee the sender pays per unit of gas

Gas is a reference to the computation required to process the transaction by a miner. Users have to pay a fee for this computation. The `gasLimit` and `gasPrice` determine the maximum transaction fee paid to the miner (more on gas on later subsections).

2.1.3 Gas

Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network.

Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas refers to the fee required to successfully conduct a transaction on Ethereum.

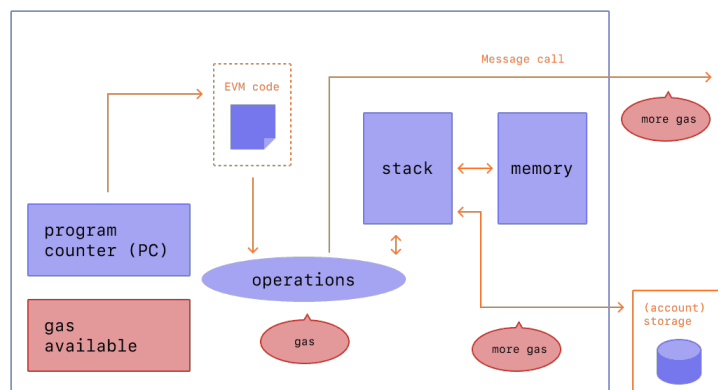


Figure 2.4: Diagram of the role of gas in Ethereum [13]

In essence, gas fees are paid in Ethereum’s native currency, ether (ETH). Gas prices are denoted in gwei, which itself is a denomination of ETH - each

gwei is equal to 0.000000001 ETH (10⁻⁹ ETH). For example, instead of saying that your gas costs 0.000000001 ether, you can say your gas costs 1 gwei.

Gas fees exist because they help keep the Ethereum network secure. By requiring a fee for every computation executed on the network, we prevent actors from spamming the network. In order to prevent accidental or hostile infinite loops or other computational wastage in code, each transaction is required to set a limit to how many computational steps of code execution it can use. The fundamental unit of computation is "gas".

2.1.4 Consensus mechanisms

When it comes to blockchains like Ethereum, which are in essence distributed databases, the nodes of the network must be able to reach agreement on the current state of the system. This is achieved using consensus mechanisms.

Consensus mechanisms (also known as consensus protocols or consensus algorithms) allow distributed systems (networks of computers) to work together and stay secure.

For decades, these mechanisms have been used to establish consensus among database nodes, application servers, and other enterprise infrastructure. In recent years, new consensus protocols have been invented to allow Cryptoeconomics systems, such as Ethereum, to agree on the state of the network.

A consensus mechanism in a cryptoeconomic system also helps prevent certain kinds of economic attacks. In theory, an attacker can compromise consensus by controlling 51% of the network. Consensus mechanisms are designed to make this "51% attack" unfeasible. Different mechanisms are engineered to solve this security problem differently.

Types of consensus mechanisms

Proof of work Proof-of-work is done by miners, who compete to create new blocks full of processed transactions. The winner shares the new block with the rest of the network and earns some freshly minted ETH. The race is won by whoever's computer can solve a math puzzle fastest – this produces the cryptographic link between the current block and the block that went before. Solving this puzzle is the work in "proof of work".

Proof of stake Proof-of-stake is done by validators who have staked ETH to participate in the system. A validator is chosen at random to create new blocks, share them with the network and earn rewards. Instead

of needing to do intense computational work, you simply need to have staked your ETH in the network. This is what incentivises healthy network behaviour.

2.1.5 Dapps

A dapp has its backend code running on a decentralized peer-to-peer network. Contrast this with an app where the backend code is running on centralized servers.

A dapp can have frontend code and user interfaces written in any language (just like an app) that can make calls to its backend. Furthermore, its frontend can be hosted on decentralized storage.

Decentralized means they are independent, and no one can control them as a group.

Deterministic they perform the same function irrespective of the environment they are executed.

Turing complete which means given the required resources, the dapp can perform any action.

Isolated which means they are executed in a virtual environment known as Ethereum Virtual Machine so that if the smart contract happens to have a bug, it won't hamper the normal functioning of the blockchain network.

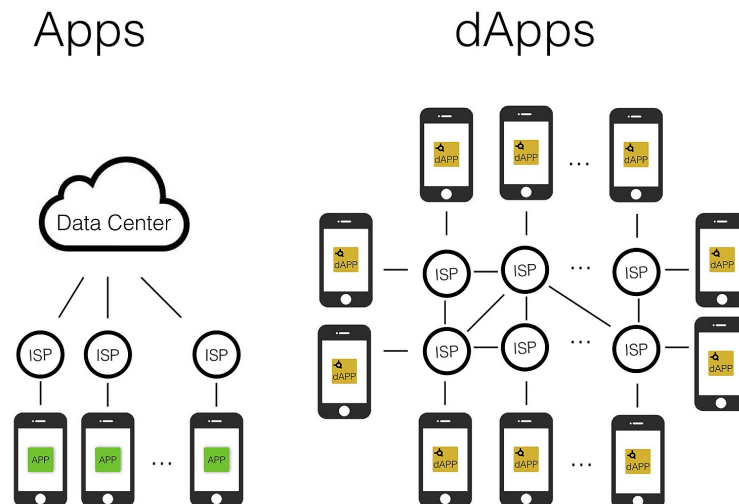


Figure 2.5: Diagram of centralized and decentralized applications topology

2.1.6 Smart contracts

A smart contract is simply put a program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain. They run exactly as programmed. Once they are deployed on the network they can't be changed. Dapps can be decentralized because they are controlled by the logic written into the contract, not an individual or company.

Smart contracts are a type of Ethereum account. This means they have a balance and they can send transactions over the network. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code.

Ethereum has developer-friendly languages for writing smart contracts:

- Solidity
- Vyper

2.2 Web Applications

A web application (or web app) is application software that runs on a web server, unlike computer-based software programs that are run locally on the operating system (OS) of the device. Web applications are accessed by the user through a web browser with an active network connection. These applications are programmed using a client-server modeled structure—the user ("client") is provided services through an off-site server that is hosted by a third-party. Examples of commonly-used web applications include: web-mail, online retail sales, online banking, and online auctions.

2.2.1 Structure

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to an n-tiered approach by nature.[3] Though many variations are possible, the most common structure is the three-tiered application.[9] In its most common form, the three tiers are called presentation, application and storage, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python or Ruby on Rails) is the middle tier (application logic),

and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

For more complex applications, a 3-tier solution may fall short, and it may be beneficial to use an n-tiered approach, where the greatest benefit is breaking the business logic, which resides on the application tier, into a more fine-grained model.[9] Another benefit may be adding an integration tier that separates the data tier from the rest of tiers by providing an easy-to-use interface to access the data. For example, the client data would be accessed by calling a "list_clients()" function instead of making an SQL query directly against the client table on the database. This allows the underlying database to be replaced without making any change to the other tiers.

There are some who view a web application as a two-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server. The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both.[9] While this increases the scalability of the applications and separates the display and the database, it still doesn't allow for true specialization of layers, so most applications will outgrow this model.

2.2.2 Business use

An emerging strategy for application software companies is to provide web access to software previously distributed as local applications. Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process. This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning can be more effective and less disruptive in the long run.

Cloud computing model web applications are software as a service (SaaS).

There are business applications provided as SaaS for enterprises for a fixed or usage-dependent fee. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface[14].

2.2.3 Development

Writing web applications is often simplified by the use of web application framework. These frameworks facilitate rapid application development by allowing a development team to focus on the parts of their application which are unique to their goals without having to resolve common development issues such as user management. Many of the frameworks in use are open-source software.

The use of web application frameworks can often reduce the number of errors in a program, both by making the code simpler, and by allowing one team to concentrate on the framework while another focuses on a specified use case. In applications which are exposed to constant hacking attempts on the Internet, security-related problems can be caused by errors in the program. Frameworks can also promote the use of best practices such as GET after POST.

In addition, there is potential for the development of applications on Internet operating systems, although currently there are not many viable platforms that fit this model[6].

Chapter 3

Implementation

In this chapter, the process of implementation will be covered, starting by the architecture and the tools to the results and how everything fits together.

3.1 Tools

3.1.1 React JS

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library[11] for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies.[10] React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

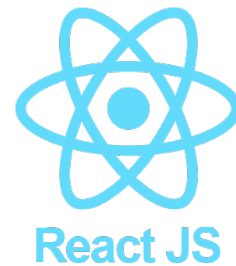


Figure 3.1: React JS framework logo

3.1.2 Web3.js

There are a few different aspects to developing blockchain applications with Ethereum:

- **Smart contract development** - writing code that gets deployed to the blockchain with the Solidity programming language.
- **Developing websites or clients that interact with the blockchain** - writing code that reads and writes data from the blockchain with smart contracts.

Web3.js enables you to fulfill the second responsibility: developing clients that interact with The Ethereum Blockchain. It is a collection of libraries that allow you to perform actions like sending Ether from one account to another, read and write data from smart contracts, create smart contracts, and so much more.



Figure 3.2: Web3 logo

Web3.js communicates to The Ethereum Blockchain with JSON RPC, which stands for "Remote Procedure Call" protocol. Ethereum is a peer-to-peer network of nodes that stores a copy of all the data and code on the blockchain. Web3.js allows us to make requests to an individual Ethereum node with JSON RPC in order to read and write data to the network[16].

3.1.3 Ganache

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.



Ganache comes in two flavors: a UI and CLI. Ganache UI is a desktop application supporting both Ethereum and Corda technology. The command-line tool, ganache-cli (formerly known as the TestRPC), is available for Ethereum development. Prefer using the command-line? This documentation will focus only on the UI flavor of Ganache. Please see the Ganache CLI Readme for command-line documentation.

Figure 3.3: Ganache Logo

All versions of Ganache are available for Windows, Mac, and Linux[7].

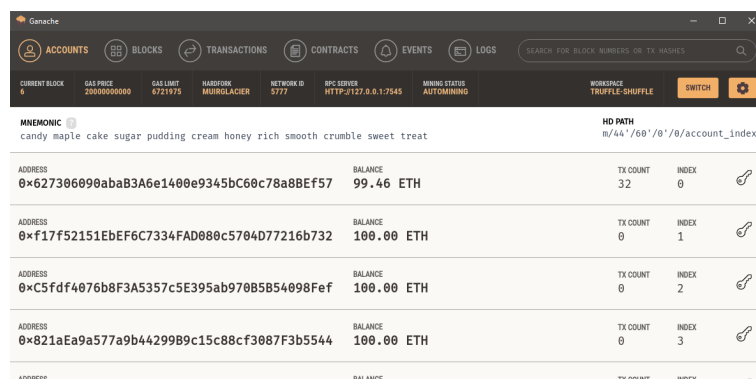


Figure 3.4: Ganache GUI

3.1.4 MetaMask

MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain.[18] It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications.



Figure 3.5:
MetaMask Logo

MetaMask allows users to store and manage account keys, broadcast transactions, send and receive Ethereum-based cryptocurrencies and tokens, and securely connect to decentralized applications through a compatible web browser or the mobile app's built-in browser.[3][4]

The application includes an integrated service for exchanging Ethereum tokens by aggregating several decentralized exchanges (DEXs) to find the best exchange rate. This feature, branded as MetaMask Swaps, charges a service fee of 0.875% of the transaction amount[19].

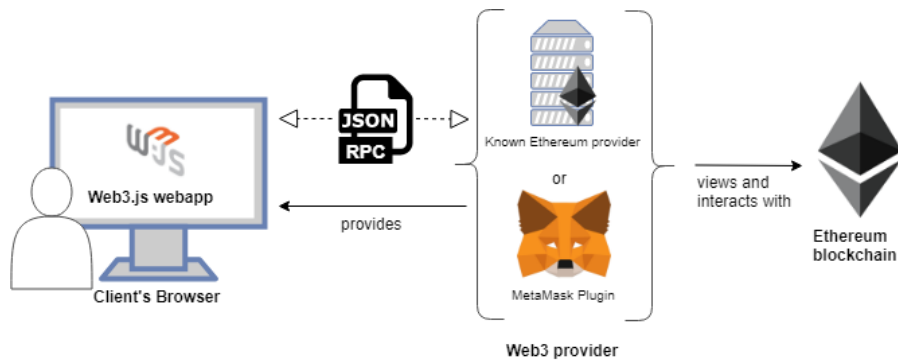


Figure 3.6: the architecture of a client application interaction with a blockchain network using Web3 and MetaMask

3.2 System Design

In this section, we introduce our proposed voting system that aims to solve some of the barriers that traditional voting systems have.

3.2.1 Architecture

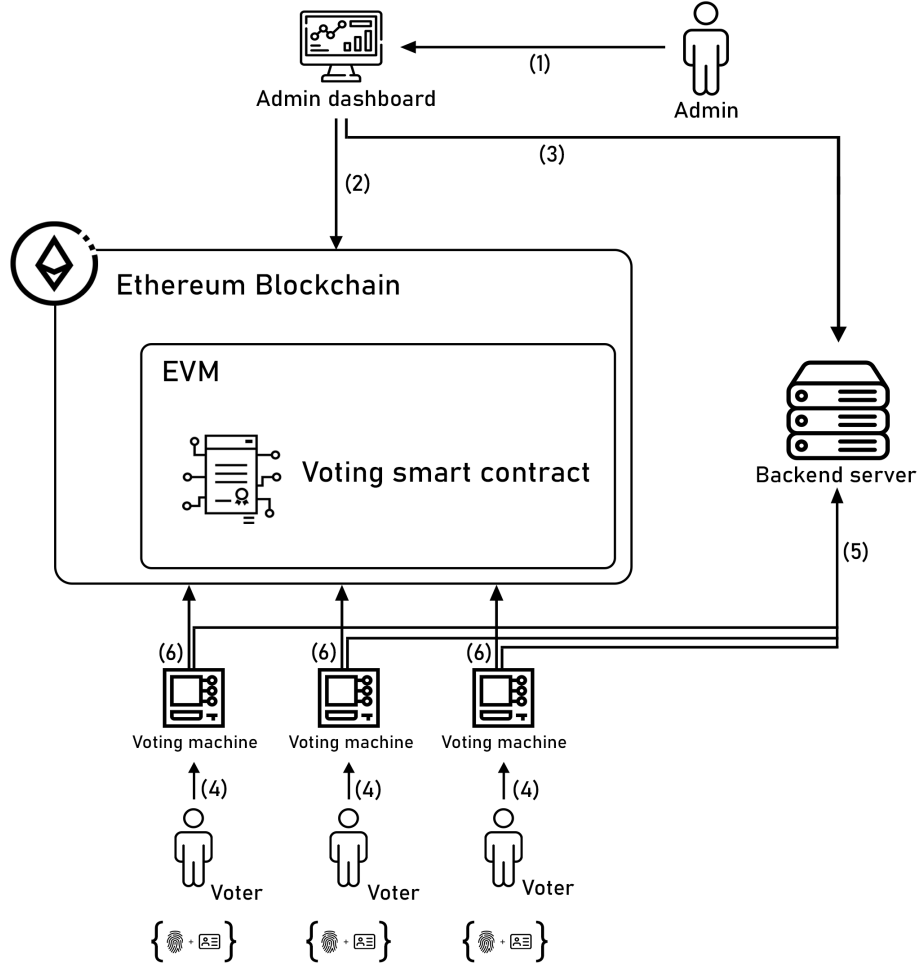


Figure 3.7: High level architecture of the proposed system and the interaction between its components

- (1) The admin operating the dashboard launches and monitors voting events
- (2) Essential data (candidates name and id, event title, start & finish dates) are saved in the blockchain along with the list of authorized accounts that can interact with the smart contract.
- (3) Secondary data that is of less importance is saved to the regular (centralized) backend server.

- (4) Voters authenticate themselves to the polling machines (using biometric means) and cast their vote for a candidate of their choosing
- (5) Voting machines use the backend server to verify the identity of the voter and to fetch secondary data required for the displaying of candidates
- (6) Voting machines saves the submitted choice to the blockchain by launching a transaction that increments the vote count of the chosen candidate

3.2.2 Diagrams

Admin sequence diagram

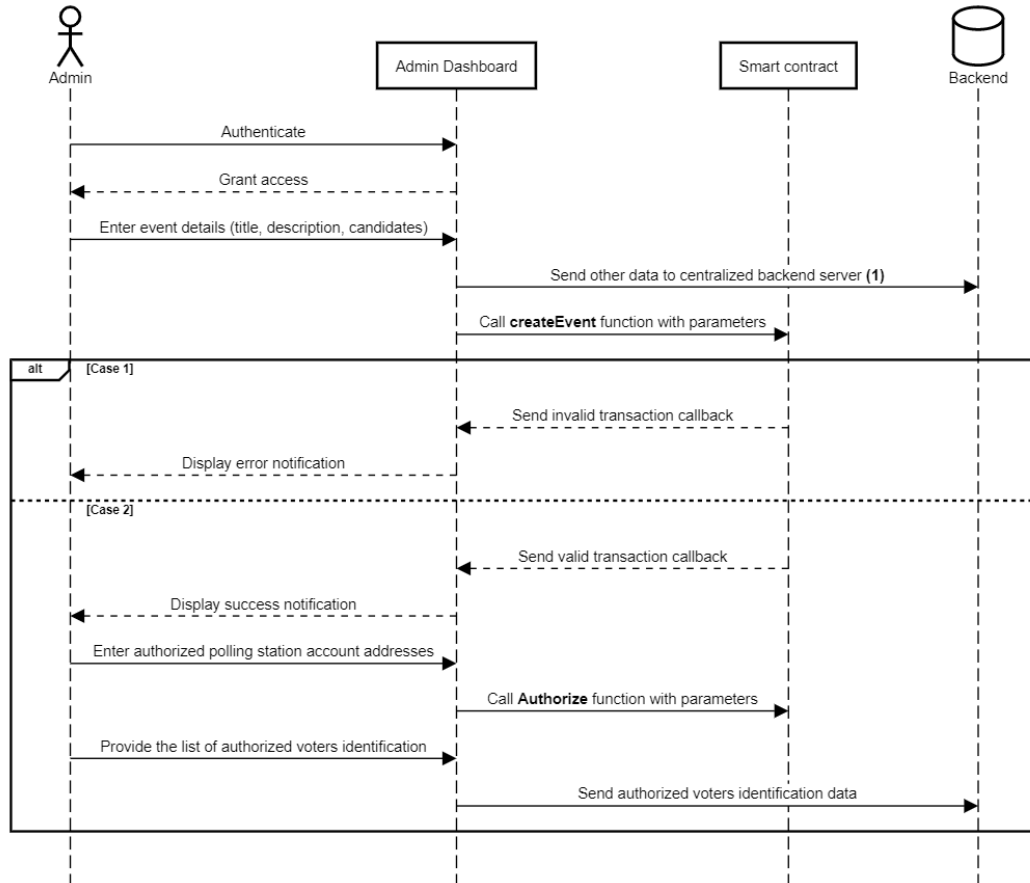


Figure 3.8: Sequence diagram of the process of creating and launching a new voting event

Launching an election requires that the administrator get access to the administration dashboard which has its access restricted by digital or physical boundaries or both, using the dashboard the administrator accesses the event creation form, in which he fills the information for the event. The information provided will be divided on the basis of their level of criticality, for example, candidates' pictures and lengthy descriptions are deemed to be less critical to the vote-counting so they're stored in a regular backend server. Also, information required for the authentication of voters is also stored on the regular backend servers.

Next, the administrator is tasked with providing the account addresses of the ethereum accounts running on each voting machine, this list will be the list of the ethereum account addresses allowed to interact with the smart contract and alter the vote count of any candidates by incrementing it.

Voter sequence diagram

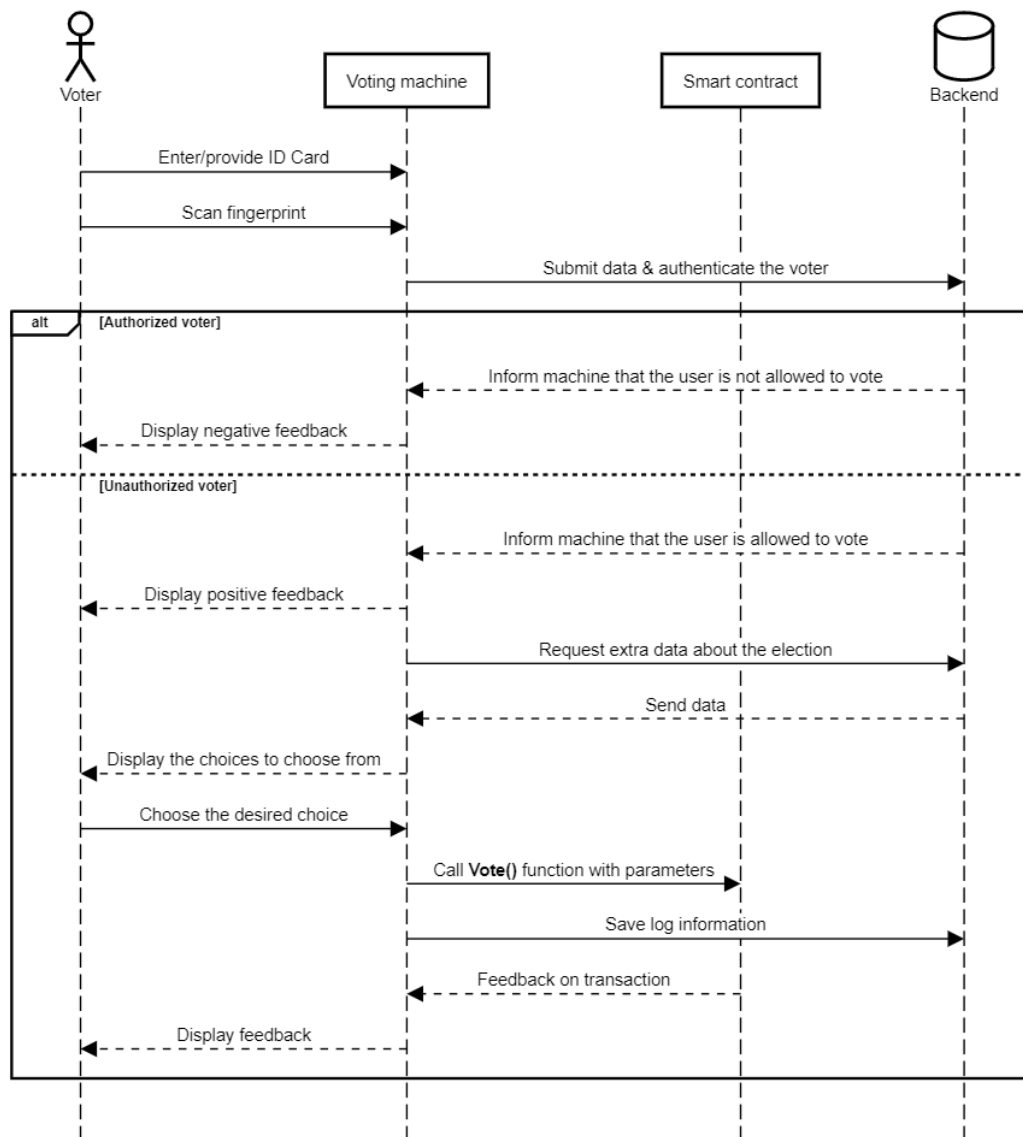


Figure 3.9: Sequence diagram of the process of casting a vote

The process of casting a vote is initiated when a voter first enters a polling station, then proceeds to a vacant voting machine, the machine should be equipped with an ID Card reader and fingerprint scanner to conduct a biometric identity check. the information obtained will be checked against the records saved on the backend server. In case the person is identified as an authorized voter; meeting whatever the requirements listed by the entity organizing the voting event (for example an entity can enforce one single vote per household or set the minimum age for participation to 50), a list of choices will be displayed to the voter to choose from in an intuitive manner after the voter makes his choice and submits it, the voting machine performance an ethereum transaction to the smart contract by calling the appropriate function that will result in incrementing the vote count of that particular candidate, also log information about the vote will be saved to the backend server.

3.3 Results

To validate the proposed system, we implemented various components of the proof-of-concept using various technologies highlighted in previous sections, in this section we will go through the results that were obtained by this investigatory effort.

3.3.1 Admin dashboard

The admin dashboard is a web application developed using React JS framework and the UI components library Shards[12], it serves as the starting point of every election event. The admin is impelled to use this unique platform to create, launch and monitor a voting event.

The access to this platform should be very restricted, as well as the ethereum account associated with it, being conceptualized to be the sole orchestrator of the whole event, this application (and account) has various privileges granted to from the deployed smart contracts, things like creating an event, terminating an event, granting voting authorizations and viewing results.

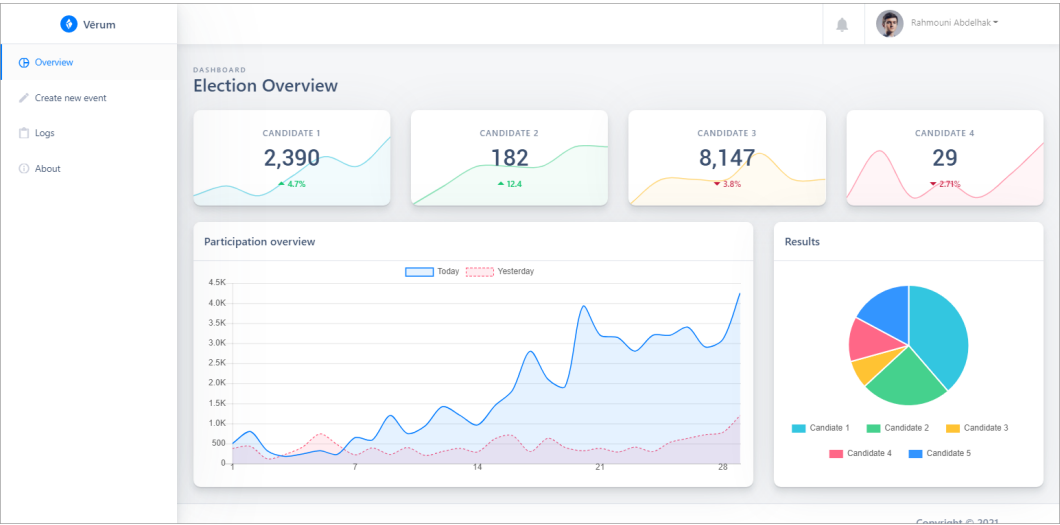


Figure 3.10: Election overview tab of the admin dashboard

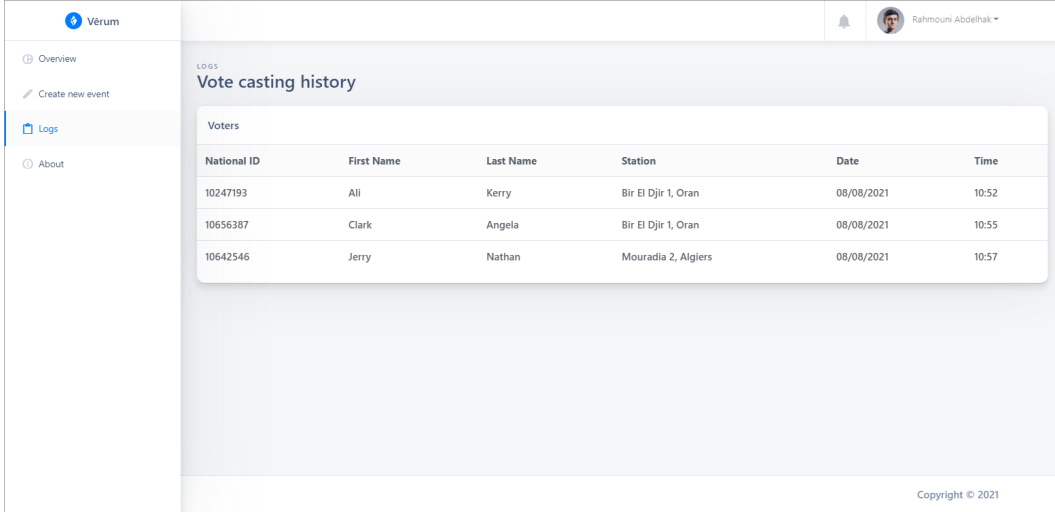
The election overview tab is the main tab where the instant monitoring of ongoing election event is done, using various forms of charts and indicators an admin can easily stay up to date with everything occurring on the ongoing event.

The screenshot shows the 'Add New Event' form in the 'EVENTS CONFIGURATION' section. The form includes fields for 'Event Title', 'Description', and 'Candidates Details' (with sub-fields for Candidate 1 through Candidate 4). There are also 'Dates' fields. A calendar widget for July 2021 is open, showing the 7th as the selected date. The dashboard layout is consistent with Figure 3.10, featuring the same sidebar and top navigation bar.

Figure 3.11: Event creation tab of the admin dashboard

Event creation tab holds an event creation form where the admin is pro-

vided with inputs fields to fill, information like the event title, description and beginning and end dates along with candidates details are all submitted through this form.



The screenshot shows an admin dashboard with a sidebar on the left containing links for 'Overview', 'Create new event', 'Logs' (highlighted), and 'About'. The main content area is titled 'Vote casting history' and contains a table with the following data:

National ID	First Name	Last Name	Station	Date	Time
10247193	Ali	Kerry	Bir El Djir 1, Oran	08/08/2021	10:52
10656387	Clark	Angela	Bir El Djir 1, Oran	08/08/2021	10:55
10642546	Jerry	Nathan	Mouradia 2, Algiers	08/08/2021	10:57

The bottom right corner of the dashboard displays 'Copyright © 2021'.

Figure 3.12: Logs tab of the admin dashboard

The logs tab presents a detailed view of the vote casting process history, unlike the election overview tab, this part of the application does not obtain its data from the blockchain ledger, but from the centralized backend that stores the log information of voters separately from the content of their vote.

3.3.2 End-user (voter) application

The voter application lives on the voting machine, although for the sake of our proof-of-concept it was built as a web application, we believe that in a real-world scenario a desktop (native) application with touch screen capabilities would serve the purpose very well.

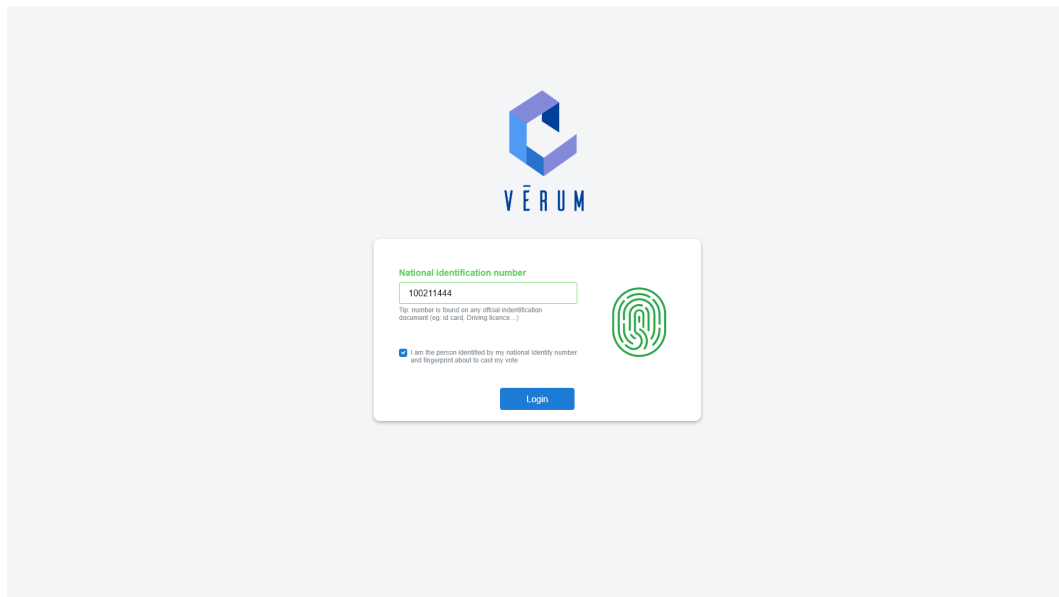


Figure 3.13: Voter application authentication screen

The voter is first faced with an authentication screen prompting her to provide her national identification number (although in a real-world scenario we would opt for electronic ID scanners) and to also scan her fingerprint, in case the voter is an authorized voter that hasn't, for example, voted before that application allows the user to proceed.

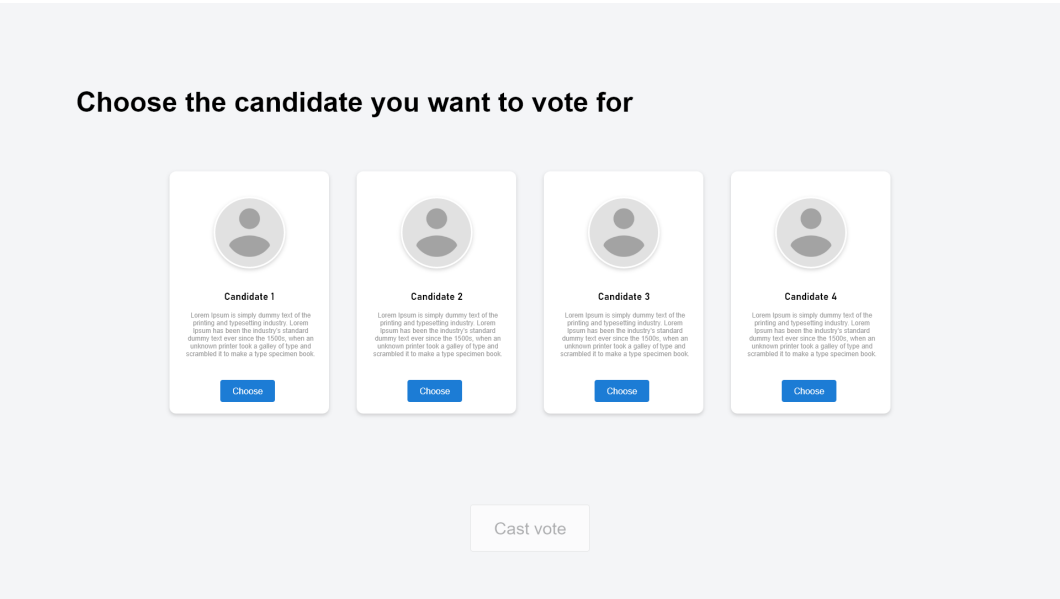


Figure 3.14: Choices presentation screen

Next, the voter is presented with the choices to pick from.

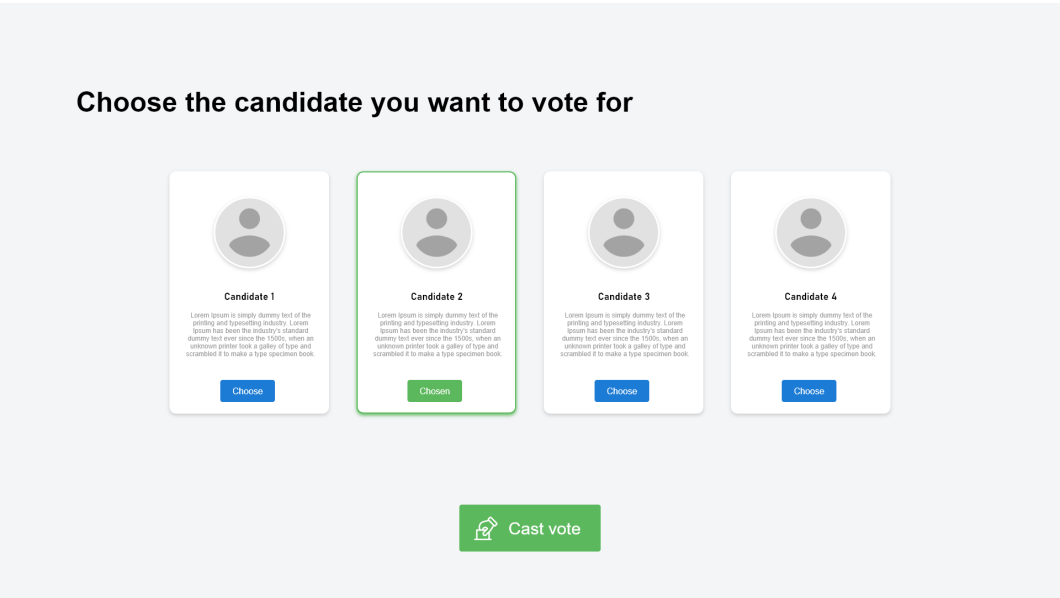


Figure 3.15: Choice submitting screen

After a choice is selected, a submit button is enabled.

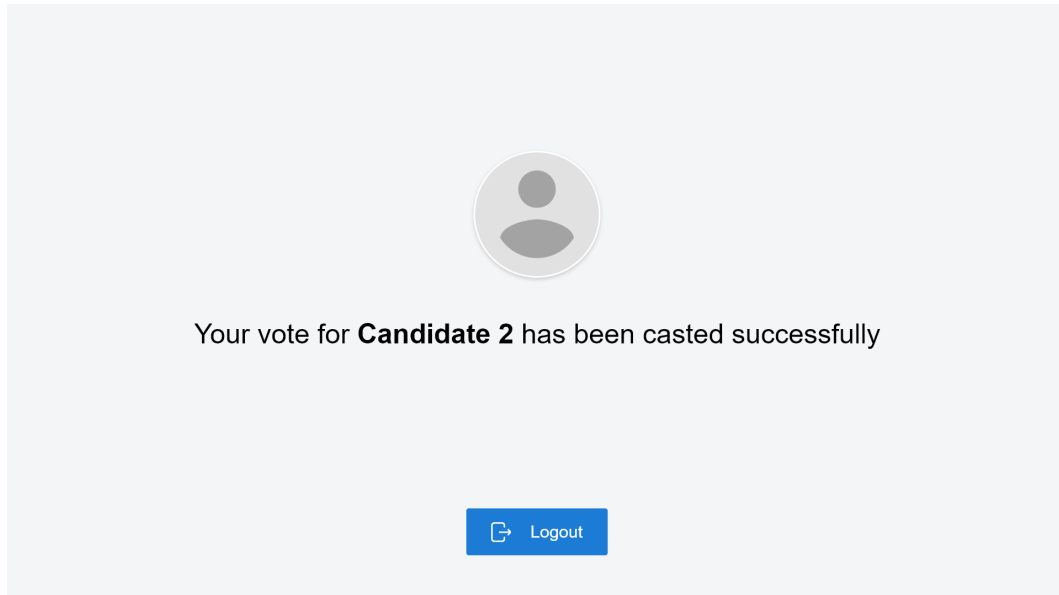


Figure 3.16: Vote casting feedback screen

Lastly, a feedback screen is displayed, then the user is prompted to log out instantly at the end of the voting process or it will log out automatically after a certain amount of time.

3.3.3 Smart contract

```

pragma solidity ^0.4.0;

contract verumElection {

    struct Candidate {
        string name;
        uint voteCount;
    }

    struct Station {
        string name;
        bool authorized;
    }

    address public owner;
    bool public electionGoing;
    Candidate[4] public candidates;
    mapping(address => Station) public stations;

    event ElectionResult(string name, uint voteCount);

    modifier onlyOwner {
        require(msg.sender == owner, "Only owner can call this function");
        _;
    }

    modifier onlyAuthorized {
        require(stations[msg.sender].authorized, "Only authorized polling stations (ETH accounts) can cast votes");
        _;
    }

    modifier onlyWhenFree {
        require(!electionGoing, "Only when there's no election already running");
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    function createEvent(string _can1, string _can2, string _can3, string _can4) public {
        electionGoing = true;

        candidates[0].name = _can1; candidates[0].voteCount = 0;
        candidates[1].name = _can2; candidates[1].voteCount = 0;
        candidates[2].name = _can3; candidates[2].voteCount = 0;
        candidates[3].name = _can4; candidates[3].voteCount = 0;
    }

    function authorize(string _name, address _address) onlyOwner public {
        require(!stations[_address].authorized);
        stations[_address].name = _name;
        stations[_address].authorized = true;
    }

    function vote(uint voteIndex) onlyAuthorized public {
        require(electionGoing);

        candidates[voteIndex].voteCount++;
    }

    function currentResult() public view returns (uint[4]) {
        uint[4] memory result;
        for(uint i=0; i<4; i++) {
            result[i] = candidates[i].voteCount;
        }
        return result;
    }

    function terminateEvent() onlyOwner public {
        require(msg.sender == owner);
        require(electionGoing);
        electionGoing = false;

        for(uint i=0; i < candidates.length; i++) {
            emit ElectionResult(candidates[i].name, candidates[i].voteCount);
        }
    }
}

```

Figure 3.17: Voting smart contract, written in solidity language

The smart contract showed in figure 3.17 implements our voting contract.

The idea is to create and deploy a smart contract using the account intended to be of the admin, then using the various functions defined the admin can interact with the smart contract to create a voting event and input information about candidates and whatnot.

Structres

Structs are custom defined types that can group several variables

Candidate is the definition of a struct type representing the information that will stored about a single candidate on the smart contract's state.

Station represents the information that will be stored about a single polling station or (a single voting machine, in case each voting machine has a separate ethereum account and address.

State variables

State variables are variables whose values are permanently stored in contract storage.

owner a state variable of type address that holds the addresses of the account that owns the smart contract, being the admin, the holder of that ethereum address has extra privileges when interacting with the smart contract.

electionGoing a state variable of type boolean that holds either true when an election event is taking place or false when there's none.

candidates an array of the struct type predefined Candidate that holds the list of choices a voter needs to choose from.

stations an array of the struct type predefined Station that holds the list of polling stations allowed to cast a vote in this ongoing voting event.

Event

Solidity events give an abstraction on top of the EVM's logging functionality. Applications can subscribe and listen to these events through the RPC interface of an Ethereum client.

ElectionResult An event that is triggered once a voting event is over and it sends back the final results.

Modifiers

Modifiers can be used to easily change the behavior of functions by executing a set of instructions prior to or after executing a function.

onlyOwner checks whether the account calling the function is the account of the contract owner, and only executes the function if it is the case.

onlyAuthorized checks whether the account calling the function has its address listed among the authorized list of accounts and only executes the function if it is the case.

onlyWhenFree checks if there's an ongoing event prior to executing the function, and only allow for the execution of the function if there's no events taking place.

Constructor

A constructor is an optional function declared with the *constructor* keyword which is executed upon contract creation, and where you can run contract initialisation code.

Our constructor executes one instruction, assigning the address of the account that deployed the contract to the state variable *owner* and granting it the admin privileges.

Functions

A function is a block of organized, reusable code that is used to perform a single, related action.

createEvent is a function that takes candidates' names as parameters and initiates or reinitiate the candidates' array and also set the *electionGoing* variable to true

authorize is a function that admin uses to add a list of account address to the *stations* array

vote is a function that is called by the voting machine to cast a vote for a particular candidate, by increment his *voteCount*

currentResult is a function that returns the current vote count of every candidate and is used by the admin to monitor the current ongoing event.

terminateEvent is a function that is used by the admin to terminate an ongoing event and emit the *ElectionResult* event.

3.3.4 System properties

Chapter 4

Conclusion and perspectives

Ecological aspects

Financial aspects

Conclusion

Perspectives

Bibliography

- [1] Article LO274 - Code électoral - Légifrance.
https://www.legifrance.gouv.fr/codes/article_lc/LEGIARTI000006353645.
- [2] Cost to Build a Hospital — Hospital Construction Cost.
<https://www.fixr.com/costs/build-hospital>.
- [3] Cybersecurity Expert Warns of Increasing Vulnerabilities in Devices — Observer. <https://observer.com/2019/09/cybersecurity-expert-asaf-ashkenazi-device-vulnerability-hacking/>.
- [4] Estimated Cost of the 43rd General Election – Elections Canada.
<https://www.elections.ca/content.aspx?section=res&dir=rep/off/cou&document=index43&>
- [5] Ethereum development documentation — ethereum.org.
<https://ethereum.org/en/developers/docs/>.
- [6] Framework - DocForge Programming Wiki.
<https://web.archive.org/web/20100327162513/http://docforge.com/wiki/Framework>.
- [7] Ganache — Overview — Documentation — Truffle Suite.
<https://www.trufflesuite.com/docs/ganache/overview>.
- [8] History Of Elections. <https://www.duvalelections.com/General-Information/Learn-About-Elections/History-Of-Elections>.
- [9] Make an N-tier architecture And give Stylish effect with Ajax & Javascript: Benefits of using the n-tiered approach for web applications. <http://krunal-ajax-javascript.blogspot.com/2008/09/benefits-of-using-n-tiered-approach-for.html>.
- [10] React: Making faster, smoother UIs for data-driven Web apps — InfoWorld. <https://www.infoworld.com/article/2608181/react-making-faster-smoother-uis-for-data-driven-web-apps.html>.

-
- [11] React – A JavaScript library for building user interfaces.
<https://reactjs.org/>.
 - [12] Shards React. <https://designrevision.com/docs/shards-react/getting-started>.
 - [13] Takenobu-hs/ethereum-evm-illustrated: Ethereum EVM illustrated.
<https://github.com/takenobu-hs/ethereum-evm-illustrated>.
 - [14] Top Tips for Secure App Development — Dell.
<https://web.archive.org/web/20120522022522/http://content.dell.com/us/en/enterprise/d/business/secure-app-development.aspx>.
 - [15] Marc Andreessen. Why Bitcoin Matters.
<https://dealbook.nytimes.com/2014/01/21/why-bitcoin-matters/>, 1390323270.
 - [16] Gregory McCubbin. Intro to Web3.js · Ethereum Blockchain Developer Crash Course. <https://www.dappuniversity.com/articles/web3-js-intro>.
 - [17] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. page 9.
 - [18] Stan Schroeder. Crypto wallet MetaMask finally launches on iOS and Android, and it supports Apple Pay.
<https://mashable.com/article/metamask-ios-android>, September 2020.
 - [19] Stan Schroeder. Crypto wallet MetaMask now lets you swap tokens on your phone. <https://mashable.com/article/metamask-swaps>, March 2021.